View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Elsevier - Publisher Connector

Procedia Computer Science

Volume 29, 2014, Pages 599–613

ICCS 2014. 14th International Conference on Computational Science

HP-DAEMON: *H*igh *P*erformance *D*istributed *A*daptive *E*nergy-efficient *M*atrix-multiplicati*ON*

Li Tan[1], Longxiang Chen[1], Zizhong Chen[1], Ziliang Zong[2], Rong Ge[3], and Dong Li[4]

[1] University of California, Riverside, California, U.S.A.
{ltan003, lchen060, chen}@cs.ucr.edu
[2] Texas State University-San Marcos, Texas, U.S.A.
zz11@txstate.edu
[3] Marquette University, Milwaukee, Wisconsin, U.S.A.
rong.ge@marquette.edu
[4] Oak Ridge National Laboratory, Oak Ridge, Tennessee, U.S.A.
lid1@ornl.gov

Abstract

The demands of improving energy efficiency for high performance scientific applications arise crucially nowadays. Software-controlled hardware solutions directed by Dynamic Voltage and Frequency Scaling (DVFS) have shown their effectiveness extensively. Although DVFS is beneficial to green computing, introducing DVFS itself can incur non-negligible overhead, if there exist a large number of frequency switches issued by DVFS. In this paper, we propose a strategy to achieve the optimal energy savings for distributed matrix multiplication via algorithmically trading more computation and communication at a time adaptively with user-specified memory costs for less DVFS switches, which saves 7.5% more energy on average than a classic strategy. Moreover, we leverage a high performance communication scheme for fully exploiting network bandwidth via pipeline broadcast. Overall, the integrated approach achieves substantial energy savings (up to 51.4%) and performance gain (28.6% on average) compared to ScaLAPACK pdgemm() on a cluster with an Ethernet switch, and outperforms ScaLAPACK and DPLASMA pdgemm() respectively by 33.3% and 32.7% on average on a cluster with an Infiniband switch.

Keywords: energy, DVFS, adaptive, memory-aware, performance, binomial tree broadcast, pipeline broadcast, ScaLAPACK, DPLASMA.

1 Introduction

With the trend of increasing number of interconnected processors providing the unprecedented capability for large-scale computation, despite exploiting parallelism for boosting performance of applications running on high performance computing systems, the demands of improving their energy efficiency arise crucially, which motivates holistic approaches from hardware to software.

Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2014

© The Authors. Published by Elsevier B.V.
doi:10.1016/j.procs.2014.05.054

Software-controlled hardware solutions [14] [10] [21] [20] of improving energy efficiency for high performance applications have been recognized as effective potential approaches, which leverage different forms of slack in terms of non-overlapped latency [23] to save energy.

For applications running on distributed-memory architectures, Dynamic Voltage and Frequency Scaling (DVFS) [24] has been leveraged extensively to save energy for components such as CPU, GPU, and memory, where the performance of the components is modified by altering its supply voltage and operating frequency. Reduction on supply voltage generally results in decrease of operating frequency as a consequence, and vice versa. Given the assumption that dynamic power consumption P of a CMOS-based processor is proportional to product of working frequency f and square of supply voltage V, i.e., $P \propto fV^2$ [19] [12], and also existing work that indicates energy costs on CPU dominate the total system energy consumption [11], DVFS is deemed an effective software-based dynamic technique to reduce energy consumption on a high performance computing system. For instance, DVFS can be leveraged to reduce CPU frequency if the current operation is not CPU-bound. In other words, execution time of the operation will barely increase if CPU frequency is scaled down. CPU frequency is kept at the highest scale if performance of the operation is harmed by decreasing CPU frequency. Energy savings can thus be achieved due to lower CPU frequency as wells as supply voltage with negligible performance loss. As a fundamental component of most numerical linear algebra algorithms [8] employed in high performance scientific computing, state-of-the-art algorithms of matrix multiplication on a distributed-memory computing system perform alternating matrix broadcast and matrix multiplication on local computing nodes with a local view [7]. Given that the communication in distributed matrix multiplication is not bound by CPU frequency while the computation is, one classic way to achieve energy efficiency for distributed matrix multiplication is to set CPU frequency to low for broadcast while set it back to high for multiplication [6] [17]. In general, considerable energy savings can be achieved from the low-power communication phase.

Although employing DVFS is beneficial to saving energy via software-controlled power-aware execution, introducing DVFS itself can cost non-negligible energy and time overhead from two aspects. Firstly, using DVFS in our approach is via dynamically modifying CPU frequency configuration files that are essentially in-memory temporary system files, for setting up appropriate frequencies at OS level. It incurs considerable memory access overhead if there exist a large number of such virtual file read and write operations for switching CPU frequency. Secondly, CPU frequency (a.k.a., gear [10]) transition latency required for taking effect (on average 100μs for AMD Athlon processors and 38μs for AMD Opteron 2380 processors employed in this work) results in additional energy costs, since a processor has to stay in use of the old frequency while switching to the newly-set frequency is not complete. We need to either minimize the time spent on memory accesses for switching frequency, i.e., the latency required for changing the gears successfully, or reduce the number of frequency switches to save energy. It is challenging to reduce the first type of time costs, since it depends on hardware-related factors such as memory accessing rate and gear transition latency. Alternatively, a software-controlled energy efficient DVFS scheduling strategy to reduce frequency switches is thus desirable.

Numerous efforts have been conducted on devising energy efficient DVFS-directed solutions, but few of them concern possible non-negligible overhead incurred from employing DVFS. In this paper, we propose an adaptive DVFS scheduling strategy with a high performance communication scheme via pipeline broadcast to achieve the optimal energy and performance efficiency for distributed matrix multiplication, named *HP-DAEMON*. Firstly, we propose a memory-aware mechanism to reduce DVFS overhead, which adaptively limits the number of frequency switches by grouping communication and computation respectively, at the cost of memory overhead within a certain user-specified threshold. Further, we take advantage of a pipeline

broadcast scheme with tuned chunk size to boost performance of communication, with which network bandwidth is exploited thoroughly compared to binomial tree broadcast.

The rest of this paper is organized as follows. Section 2 discusses related work and section 3 introduces distributed matrix multiplication. We present an adaptive DVFS scheduling strategy in section 4 and a high performance pipeline broadcast communication scheme in section 5. We provide details of implementation and evaluation in section 6. Section 7 concludes the paper.

2 Related Work

Numerous energy saving DVFS scheduling strategies exist without considering possible non-negligible overhead on employ DVFS, including some high performance communication schemes.
DVFS Scheduling for CPU-bound Operations: Alonso *et al.* [4] leveraged DVFS to enable an energy efficient execution of LU factorization, where idle threads were set into blocked and CPU frequency of the corresponding core was lowered down to save energy with minor performance loss (up to 2.5%). Energy savings achieved were not much (up to 5%) since the approach was only applied to a shared-memory system where the slack can only result from idle CPU usage locally. Kimura *et al.* [17] employed DVFS into two distributed-memory parallel applications to allow tasks with slack to execute at an appropriate CPU frequency that does not increase the overall execution time. Energy savings up to 25% were reported with minor performance loss (as low as 1%). In their work, reducing DVFS overhead was not studied.
DVFS Scheduling for MPI Programs: Kappiah *et al.* [14] presented a dynamic system that reduces CPU frequency on nodes with less computation and more slack to use. With little performance loss, their approach was able to save energy for power-scalable clusters, where the computation load was imbalanced. Springer *et al.* [21] presented a combination of performance modeling, performance prediction, and program execution to find a near-optimal schedule of number of nodes and CPU frequency to satisfy energy costs and execution time requirements. Li *et al.* [18] proposed a strategy to improve energy efficiency for hybrid parallel applications where both shared and distributed-memory programming models (OpenMP and MPI) were employed. The key difference between our approach and these solutions is that we take the overhead on applying DVFS into account and minimize its costs to save energy.
Improving MPI Communication Performance: Chan *et al.* [5] redesigned MPI communication algorithms to achieve that one node can communicate with multiple nodes simultaneously with lower costs rather than one-to-one at a time. Faraj *et al.* [9] presented a customized system that generates efficient MPI collective communication routines via automatically-generated topology specific routines and performance tuning to achieve high performance consistently. Karwande *et al.* [16] presented an MPI prototype supporting compiled communication to improve performance of MPI communication routines, which allowed the user to manage network resources to aggressively optimize communication. Hunold *et al.* [13] proposed a mechanism that automatically selected a suitable set of blocking factors and block sizes for pdgemm() to maximize performance. Our approach differs from these techniques, since it improves MPI communication performance via highly-tuned pipeline broadcast that maximizes the slack utilization, without modifying MPI communication routines and the pdgemm() routine interface.

3 Background: Distributed Matrix Multiplication

Matrix multiplication is one fundamental operation of most numerical linear algebra algorithms for solving a system of linear equations, such as Cholesky, LU, and QR factorizations [8], where

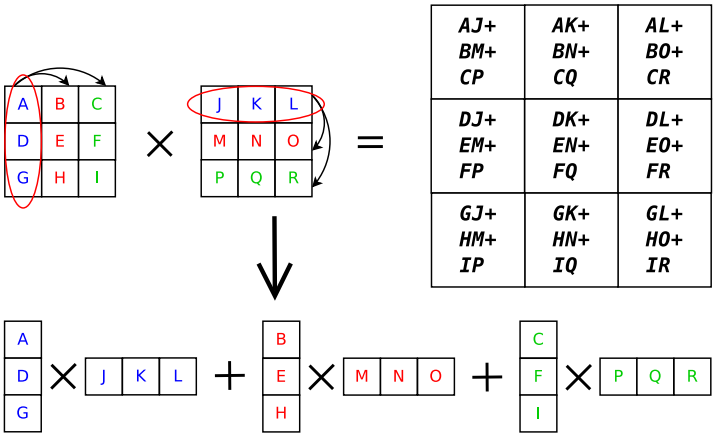


Figure 1: A Distributed Matrix Multiplication Algorithm with a Global View.

runtime percentage of matrix multiplication can be up to 92% [22]. Moreover, nowadays matrix multiplication has been widely used in many areas other than scientific computing, including computer graphics, quantum mechanics, game theory, and economics. In scientific computing, various software libraries of numerical linear algebra for distributed multi-core high performance scientific computing (ScaLAPACK [3] and DPLASMA [2]) have routines of various functionality where matrix multiplication is involved. In this paper, we aim to achieve the optimal energy and performance efficiency for distributed matrix multiplication in general. Our approach works at library level and thus serves as a cornerstone of saving energy and time for other numerical linear algebra operations where matrix multiplication is intensively employed.

3.1 Algorithmic Details

The matrix multiplication routines from ScaLAPACK/DPLASMA are essentially derived from DIMMA (Distribution-Independent Matrix Multiplication Algorithm), an advanced version of SUMMA (Scalable Universal Matrix Multiplication Algorithm) [7]. The core algorithm consists of three steps: (a) Partition the global matrix into the process grid using load balancing techniques, (b) broadcast local sub-matrices in a row-/column-wise way as a logical ring, and (c) perform local sub-matrix multiplication. Applying an optimized communication scheme, DIMMA outperforms SUMMA by eliminating slack from overlapping computation and communication effectively. Next we illustrate DIMMA using Directed Acyclic Graph (DAG) representation.

3.2 DAG Representation

In general, a task-parallel application such as distributed matrix multiplication can be partitioned into a cluster of computing nodes for parallel execution. The method of partitioning greatly affects the outcomes of energy and performance efficiency. During task partitioning, data dependencies between tasks must be respected for correctness. When the application is partitioned, data dependencies between distributed tasks can be represented using DAGs, which characterize parallel executions of tasks effectively. A formal definition of DAGs is given below:

Definition 1. Data dependencies between tasks from partitioning a task-parallel application are modeled by a Directed Acyclic Graph (DAG) $G = (V, E)$, where each node $v \in V$ denotes a task, and each directed edge $e \in E$ denotes a dynamic data dependency between the tasks.

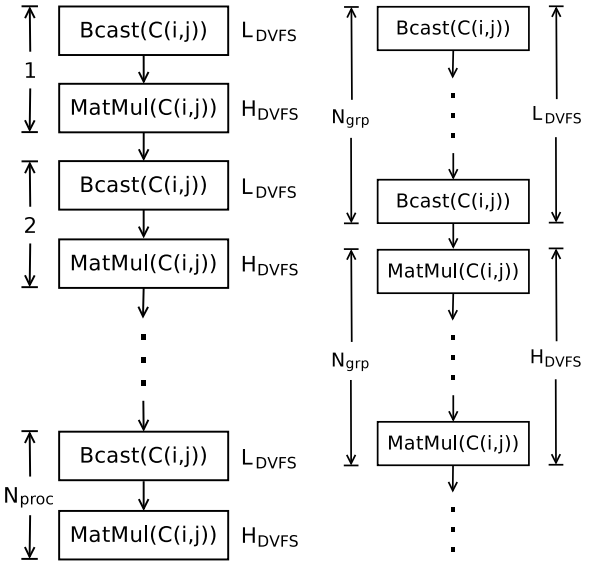


Figure 2: Matrix Multiplication DAGs with Two DVFS Scheduling Strategies.

Next we show how a task-parallel application is partitioned to achieve parallelism and represented in DAG, taking distributed matrix multiplication for example. Consider multiplying of a $m \times k$ matrix A and a $k \times n$ matrix B, to produce a $m \times n$ matrix C. For calculating a matrix element (i, j) in C, denoted as $C(i, j)$, we apply a cache efficient blocking method, where columns of A multiply rows of B to reduce cache misses, as shown in Figure 1. Recall that distributed matrix multiplication requires alternating matrix broadcast and matrix multiplication, as two DAGs shown in Figure 2. Each DAG represents an execution of calculating $C(i, j)$ with a DVFS scheduling strategy, where Figure 2 (a) gives the DVFS scheduling strategy employed in [6] [17], and the adaptive DVFS scheduling strategy proposed in this paper is shown in Figure 2 (b). Given matrices A and B, each matrix row needs to be broadcasted to other rows located in different nodes and likewise each matrix column needs to be broadcasted to other nodes, such that sub-matrices of the resulting matrix C are calculated locally and accumulated to C globally. As the strategy shown in Figure 2 (a), each broadcast step is followed by a multiplication step alternatingly until all sub-matrices of A and B involved in calculating $C(i, j)$ are broadcasted and computed, where $\mathsf{Bcast}(C(i, j))$ denotes step-wise broadcasting sub-matrices of A and B that are involved in calculating $C(i, j)$, and $\mathsf{MatMul}(C(i, j))$ denotes step-wise multiplying and accumulating these sub-matrices of A and B that are broadcasted.

4 Adaptive Memory-aware DVFS Scheduling Strategy

Next we present an adaptive memory-aware DVFS scheduling strategy (referred to as *DAEMON* henceforth) that limits the overhead on employing DVFS, at the cost of user-specified memory overhead threshold, which determines the extent of DVFS overhead. The heuristic of *DAEMON* is straightforward: Combine multiple broadcasts/multiplications as a group to reduce the number of frequency switches by DVFS, i.e., trade more computation and communication at a time with the memory cost trade-off for less DVFS switches, as shown in Figure 2

Table 1: Notation in Adaptive Memory-aware DVFS Scheduling Strategy.

N	Global matrix size in blocked distributed matrix multiplication
BS	Block size in blocked distributed matrix multiplication
H_{DVFS}	The highest CPU frequency set by DVFS
L_{DVFS}	The lowest CPU frequency set by DVFS
N_{proc}	Square root of the total number of processes in a process grid
N_{grp}	Number of broadcasts/multiplications executed at a time as a group in a process grid
S_{mem}	The total system memory size for one node
T_{mem}	A user-specified memory cost threshold for grouping, in terms of a percentage of S_{mem}
e_{DVFS}^{unit}	Energy consumption from one frequency switch
e_{DVFS}^{basic}	Energy consumption from the basic strategy employed in [6] [17]
e_{DVFS}^{daemon}	Energy consumption from *DAEMON* proposed in this paper

(b). Instead of performing a multiplication immediately after a broadcast, we keep broadcasting several times as a group, followed by corresponding multiplications as a group as well. Note that the number of broadcasts in a broadcast group must equal the number of multiplications in a multiplication group to guarantee the correctness. The number of DVFS switches is thus decreased since we only need to perform one DVFS switch for a group rather than individual broadcast/multiplication. Table 1 lists the notation used in Figure 2 and the later text.

As shown in Figure 2 (a), the basic DVFS scheduling strategy sets CPU frequency to low during broadcast while sets it back to high during matrix multiplication for energy efficiency [6] [17]. A primary defect of this strategy is that it requires two DVFS switches in one iteration, totally $2 \times N_{proc}$ DVFS switches for one process. For performance purposes in high performance computing, block-partitioned algorithms are widely adopted to maximize data reuse opportunities by reducing cache misses. In a blocked distributed matrix multiplication such as the pdgemm() routine provided by ScaLAPACK, if the basic strategy is applied, the total number of DVFS switches is $2 \times N_{proc} \times \frac{N/N_{proc}}{BS} \times N_{proc}^2$, since there are $\frac{N/N_{proc}}{BS}$ pairs of local communication and computation for each process and totally N_{proc}^2 processes in the process grid, where N/N_{proc} is the local matrix size. Given a huge number of DVFS switches, the accumulated time and energy overhead on employing DVFS can be considerable. Next we introduce details of *DAEMON* to minimize the DVFS overhead and thus achieve the optimal energy savings.

4.1 Memory-aware Grouping Mechanism

Intuitively, the heuristic of *DAEMON* for grouping broadcasts/multiplications requires for each process, keeping several sub-matrices of A and B received from broadcasts of other processes in memory for later multiplications at a time. *DAEMON* restricts the memory costs from holding matrices in memory for future computation to a certain threshold, which can be modeled as:

$$8 \times \left(\frac{N}{N_{proc}}\right)^2 \times 2 \times N_{grp} \times N_{proc} \leq S_{mem} \times T_{mem} \quad \text{subject to } 1 \leq N_{grp} \leq N_{proc} \tag{1}$$

where 8 is the number of bytes used by a double-precision floating-point number and $\frac{N}{N_{proc}}$ is the local matrix size. For each process, we need to keep totally $2 \times N_{grp}$ sub-matrices of A and B in the memory of one node for N_{grp} broadcasts and N_{grp} multiplications performed at a time, and there are N_{proc} processes for one node. Following the constraints of Equation 1, we can calculate the optimal N_{grp} from a memory cost threshold value T_{mem} for specific hardware.

Algorithm 1 *Adaptive Memory-aware DVFS Scheduling Strategy*

SetDVFS(N, N_{proc})
1: $S_{mem} \leftarrow$ GetSysMem()
2: $T_{mem} \leftarrow$ GetMemTshd()
3: $unit \leftarrow N/N_{proc}$
4: $N_{grp} \leftarrow S_{mem} \times T_{mem}/(8 \times unit^2 \times 2)$
5: $nb \leftarrow N_{proc}/N_{grp}$
6: **foreach** $i < nb$ **do**
7: **if** (IsBcast() && $freq$!= L_{DVFS}) **then**
8: SetFreq(L_{DVFS})
9: **end if**
10: **if** (IsMatMul() && $freq$!= H_{DVFS}) **then**
11: SetFreq(H_{DVFS})
12: **end if**
13: **end for**

4.2 DAEMON Algorithm

We next show how *DAEMON* reduces DVFS overhead via grouping. In accordance with Equation 1, given a memory cost threshold T_{mem} and a specific hardware configuration, the optimal N_{grp} that determines the extent of grouping can be calculated. At group level, CPU frequency is then set to low for N_{grp} times grouped broadcasts and set back to high for N_{grp} times grouped multiplications at a time, instead of being switched for individual broadcast/multiplication. Consequently, the number of CPU frequency switches are greatly decreased by *DAEMON*.

Algorithm 1 presents detailed steps of calculating N_{grp} and then employ DVFS at group level. It first calculates N_{grp} using the user-specified threshold T_{mem}, and then set frequency accordingly for grouped broadcasts/multiplications, where the number of DVFS switches is minimized. Variable $freq$ denotes current operating CPU frequency, and functions GetSysMem(), GetMemTshd(), SetFreq(), IsBcast(), and IsMatMul() were implemented to get the total system memory size, get memory cost threshold specified by the user, set specific frequencies using DVFS, and determine if the current operation is either a broadcast or a multiplication, respectively.

4.3 Energy Efficiency Analysis

DAEMON minimizes DVFS overhead by reducing the number of frequency switches, at the cost of memory overhead within a user-specified threshold T_{mem}. The optimal value of N_{grp} for minimizing DVFS overhead can be calculated from the threshold, which determines the extent of grouped broadcasts/multiplications for less frequency switches. Per Equation 1, for blocked distributed matrix multiplication, energy costs on employing DVFS in the basic strategy and in our *DAEMON* strategy shown in Figure 2 are modeled respectively, in terms of the product of unit energy costs on one DVFS switch and the number of such switches as follows:

$$e_{DVFS}^{basic} = e_{DVFS}^{unit} \times 2 \times N_{proc} \times \frac{N/N_{proc}}{BS} \times N_{proc}^2 \tag{2}$$

$$e_{DVFS}^{daemon} = e_{DVFS}^{unit} \times 2 \times \frac{N_{proc}}{N_{grp}} \times \frac{N/N_{proc}}{BS} \times N_{proc}^2 \tag{3}$$

According to Equations 2 and 3, we derive energy deflation (i.e., *ratio*) and energy savings (i.e., *difference*) from employing *DAEMON* against the basic strategy as below:

$$E_{def} = \frac{e_{DVFS}^{basic}}{e_{DVFS}^{daemon}} = N_{grp} \tag{4}$$

$$\begin{aligned} E_{sav} &= e_{DVFS}^{basic} - e_{DVFS}^{daemon} \\ &= e_{DVFS}^{unit} \times 2 \times \frac{N}{BS} \times N_{proc}^2 \times \left(1 - \frac{1}{N_{grp}}\right) \end{aligned} \tag{5}$$

From Equations 4 and 5, we can see that both E_{def} and E_{sav} greatly depend on the value of N_{grp}. The greater N_{grp} is, the more energy efficient *DAEMON* is. Following the constraints of Equation 1, we know in the best case that $N_{grp} = N_{proc}$, $E_{def} = N_{proc}$, while in the worst case that $N_{grp} = 1$, $E_{def} = 1$ as well, since *DAEMON* regresses to the basic strategy in this case. Further, we know that the energy saved from *DAEMON* can be huge given a large value of $\frac{N}{BS}$. In general, *DAEMON* is more energy efficient in contrast to the basic strategy, if $N_{grp} > 1$.

5 High Performance Communication Scheme

In addition to applying *DAEMON* to minimize DVFS overhead in distributed matrix multiplication for energy efficiency, we also aim to achieve performance efficiency and thus additional energy savings can be achieved from less execution time. Generally, performance gain of distributed matrix multiplication can be fulfilled in terms of high performance computation and communication. Given that the optimal computation implementation of local matrix multiplication routine provided by ATLAS [1] is employed, we propose a high performance communication scheme for fully exploiting network bandwidth. Specifically, since the global matrix is evenly distributed into the process grid for load balancing, we need to broadcast each matrix row/column to all other rows/columns located in different nodes individually for later performing local matrix multiplication in parallel. A high performance broadcast algorithm is thus desirable.

5.1 Binomial Tree and Pipeline Broadcast

There exist a large body of distributed broadcast algorithms for high performance communication, where binomial tree and pipeline broadcast generally outperform other algorithms for different system configurations. In the original `pdgemm()` routine from ScaLAPACK on top of different MPI implementations, different communication schemes like ring-based, binomial tree and pipeline broadcast are adopted depending on message size and other factors [3]. Table 2 lists the notation used in this section. Figure 3 (a) depicts how the binomial tree broadcast algorithm works using a simple example with a 3-round iteration on a 8-process cluster. We can see that in each round, a process sends messages in accordance with the following pattern:

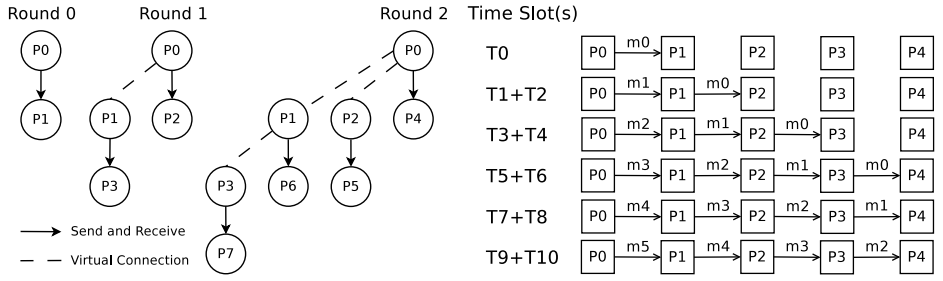
- In Round 0, process P_0 (sender) sends a message to the next available process P_1 (receiver);
- In Round j ($j > 0$), process P_i ($i \leq j,\ i = 0, 1, 2, \dots$) that is a sender/receiver in the precedent round sends a message to the next available process, until the last one receives.

In other words, in Round j, the number of senders/receivers is 2^j and thus the algorithm takes $\log P$ rounds for the P^{th} process to receive a message. The communication time complexity can be modeled as:

$$T_B = T_b \times \log P, \text{ where } T_b = T_s + \frac{S_{msg}}{BD} \tag{6}$$

Table 2: Notation in Binomial Tree and Pipeline Broadcast.

P	The total number of processes in the communication
S_{msg}	Message size in one broadcast
BD	Network bandwidth in the communication
T_B	The total time consumed by all binomial tree broadcasts
T_P	The total time consumed by all pipeline broadcasts
T_b	Time consumed by one binomial tree broadcast
T_p	Time consumed by one pipeline broadcast
T_s	Network latency of starting up a communication link
T_d	Time consumed by transmitting messages
n	Number of chunks from dividing a message



(a) Binomial Tree Broadcast (b) Pipeline Broadcast

Figure 3: Binomial Tree and Pipeline Broadcast Algorithm Illustration.

By substituting T_b, we obtain the final time complexity formula of binomial tree broadcast:

$$T_B = \left(T_s + \frac{S_{msg}}{BD}\right) \times \log P \tag{7}$$

Pipeline broadcast works in a time-sliced fashion so that different processes simultaneously broadcast different message chunks as stages in pipelining, as shown in Figure 3 (b). Assume a message in the pipeline broadcast is divided into n chunks. when the pipeline is not saturated, i.e., in the worst case, it takes $n + P - 1$ steps for the P^{th} process to receive a whole message of n chunks. We can model the time complexity of pipeline broadcast as:

$$T_P = T_p \times (n + P - 1), \text{ where } T_p = T_s + \frac{S_{msg}/n}{BD} \tag{8}$$

Similarly, substituting T_p into T_P in Equation 8 yields:

$$T_P = \left(T_s + \frac{S_{msg}/n}{BD}\right) \times (n + P - 1) \tag{9}$$

From Equations 7 and 9, despite the steps needed to receive a message, we can see that both T_B and T_P are essentially the sum of T_s and T_d. In a cluster connected by an Ethernet/Infiniband switch, T_s is of the order of magnitude of μs, so T_s is negligible when S_{msg} is comparatively large. Therefore, Equations 7 and 9 can be further simplified as follows:

$$T_B \approx \frac{S_{msg}}{BD} \times \log P \text{ and } T_P \approx \frac{S_{msg}}{BD} \times \left(1 + \frac{P-1}{n}\right) \tag{10}$$

It is clear that both T_B and T_P scale up as P increases, with fixed message size and fixed number of message chunks. However the pipeline broadcast outperforms the binomial tree broadcast with given P and message size by increasing n, since the ratio of binomial tree broadcast to pipeline broadcast approximates to $\log P$ when n is large enough and $\frac{P-1}{n}$ becomes thus negligible. We experimentally observed the communication schemes in ScaLAPACK and DPLASMA pdgemm() routines are not optimal in two clusters. Energy and time saving opportunities can be exploited by leveraging slack arising from the communication. Thus a high performance pipeline broadcast scheme with tuned chunk size according to system characteristics is desirable.

6 Implementation and Evaluation

We have implemented our high performance adaptive memory-aware DVFS scheduling strategy with highly tuned pipeline broadcast (referred to as *HP-DAEMON* in the later text) to achieve the optimal energy and performance efficiency for distributed matrix multiplication. Our implementation was accomplished by rewriting pdgemm() from ScaLAPACK [3] and DPLASMA [2], two widely used high performance and scalable numerical linear algebra libraries for distributed-memory multi-core systems nowadays. In our implementation, instead of using binomial tree broadcast for communication, we tune chunk size of pipeline broadcast to fully exploit possible slack during communication [22]. We apply the core tiling topology similarly as proposed in [15] to exploit more parallelism in communication. For achieving the maximal performance of computation, we employ the dgemm() routine provided by ATLAS [1], a linear algebra software library that automatically tunes performance according to configurations of the hardware. The rewritten pdgemm() has the same interface and is able to produce the same results as the original ScaLAPACK/DPLASMA pdgemm() routines, with total normalized differences between the range of 10^{-17} and 10^{-15} in our experiments. For comparison purposes, we also implemented the basic DVFS scheduling strategy (referred to as *Basic DVFS* later) employed in [6] [17].

Specifically, *HP-DAEMON* was implemented from two aspects as an integrated energy and performance efficient approach. Given a memory cost threshold specified by the user as a trade-off for saving energy, *HP-DAEMON* adaptively calculates N_{grp}, following the constraints of Equation 1. Then N_{grp} is applied to determine the extent of grouping, which reduces the number of DVFS switches at the cost of user-specified memory overhead. For performance efficiency, during computation, *HP-DAEMON* employs the optimal implementation of local matrix multiplication; during communication, *HP-DAEMON* leverages a non-blocking version [22] of the pipeline broadcast with tuned chunk size to maximize network bandwidth utilization.

6.1 Experimental Setup

We applied *HP-DAEMON* to five distributed matrix multiplications with different global matrix sizes to assess energy savings and performance gain achieved by our integrated approach. Experiments were performed on a small-scale cluster (HPCL) with an Ethernet switch consisting of 8 computing nodes with two Quad-core 2.5 GHz AMD Opteron 2380 processors (totalling 64 cores) and 8 GB RAM running 64-bit Linux kernel 2.6.32, and a large-scale cluster (Tardis) with an Infiniband switch consisting of 16 computing nodes with two 16-core 2.1 GHz AMD Opteron 6272 processors (totalling 512 cores) and 64 GB RAM running the same Linux kernel. All energy-related experiments were conducted only on HPCL while all performance-related experiments were performed on both clusters, since only HPCL was equipped with power sensors and meters for energy measurement. In our experiments, energy consumption was measured

using PowerPack [11], a comprehensive software and hardware framework for energy profiling and analysis of high performance systems and applications. The range of CPU frequencies on HPCL was {0.8, 1.3, 1.8, 2.5} GHz. PowerPack was deployed and running on a meter node to collect energy costs on all involved components such as CPU, memory, disk, motherboard, etc. of all 8 computing nodes within HPCL. The collected energy information was recorded into a log file in the local disk and accessed after executing these distributed matrix multiplications.

6.2 Overhead on Employing DVFS

The introduction of DVFS may incur non-negligible energy and time costs on in-memory file read/write operations and gear transitions, if fine-grained DVFS scheduling is employed as in the case of *Basic DVFS*. In order to obtain accurate DVFS overhead, we measured energy and time costs on CPU frequency switches separately from the running application on the HPCL cluster, as shown in Figure 5. We can see that both energy and time costs increase monotonically as the number of DVFS switches increases. On average it takes about 70μs for one DVFS switch to complete and take effect, and about every 10 DVFS switches incur extra one Joule energy consumption. The additional energy costs from DVFS can be considerably large if CPU frequency switches occur frequently. A smart way of reducing the number of DVFS switches like *HP-DAEMON* can thus save energy and time of running the application.

6.3 Memory Cost Trade-off from HP-DAEMON

Once specifying a memory overhead threshold using command line parameters, in the form of the original command of executing the application followed by an optional parameter "`-t`" with a percentage, "`-t 0.2`" for instance, the user is afterwards not involved for an energy saving execution, e.g., dynamically modifying the threshold. *HP-DAEMON* adaptively calculates the optimal value for grouping according to the custom threshold. Essentially the total memory overhead consists of the memory costs from the execution of the original application, and the additional memory costs from *HP-DAEMON*. Using the calculated grouping value, *HP-DAEMON* performs grouped computation/communication accordingly to minimize the number of DVFS switches. Table 3 lists N_{grp} values for five different matrices, corresponding theoretical values of extra memory costs from *HP-DAEMON* (i.e., the left hand side of Equation 1), and observed memory costs overall. For simplicity, in our implementation all calculated grouping values are rounded to multiples of 2. As Table 3 shows, the empirical observed total memory costs generally increase more than the theoretical extra memory costs from *HP-DAEMON* as N_{grp} doubles. This is because atop the original application, besides extra memory footprint for grouped broadcasts/multiplications, *HP-DAEMON* incurs more memory costs on stacks of grouped function calls involved in grouping that cannot be freed and re-allocated immediately.

6.4 Performance Gain via Tuned Pipeline Broadcast

From Equations 10, we have two inferences: (a) Performance of pipeline broadcast is strongly tied to chunk size, and (b) performance of pipeline broadcast can be better than binomial tree broadcast, since the ratio of binomial tree broadcast to pipeline broadcast is $logP$ when $\frac{P-1}{n}$ is negligible. Next we evaluate performance gain from the use of pipeline broadcast via tuning chunk size, in contrast to binomial tree broadcast, where global matrix sizes of distributed matrix multiplication on HPCL and Tardis are 10240 and 20480, individually.

In accordance with Equations 10, we can see in Figure 4, performance gain is achieved with the increase of chunk size (thus the decrease of the number of chunks) of pipeline broadcast, and

Table 3: Memory Overhead Thresholds for Different Matrices and N_{grp}.

Global Matrix Size	N_{grp}	Theoretical Additional Memory Overhead	Observed Total Memory Overhead
7680	2	3.2%	6.4%
	4	6.4%	8.8%
	8	12.8%	14.4%
10240	2	4.8%	10.4%
	4	9.6%	16.0%
	8	19.2%	25.6%
12800	2	8.0%	16.0%
	4	16.0%	24.0%
	8	32.0%	40.0%
15360	2	11.2%	23.2%
	4	22.4%	35.2%
	8	44.8%	57.6%
17920	2	16.0%	28.0%
	4	32.0%	43.2%
	8	64.0%	78.4%

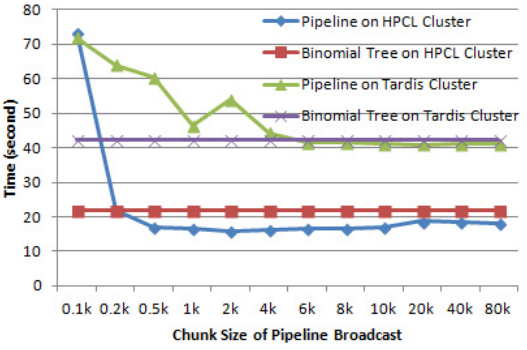


Figure 4: Performance Efficiency btw. Binomial Tree Broadcast and Pipeline Broadcast.

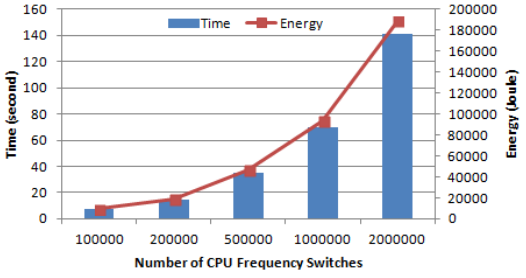


Figure 5: DVFS Energy/Time Overhead.

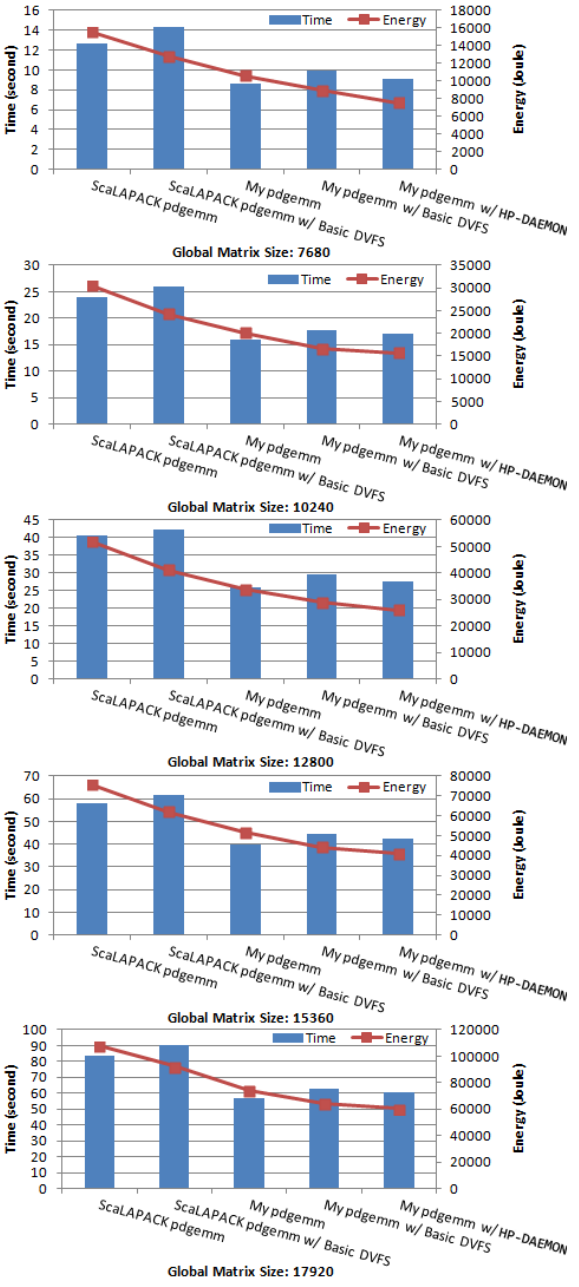


Figure 6: Energy Savings and Performance Gain on the HPCL Cluster (64-core, Ethernet).

pipeline broadcast performance converges reasonably well on both clusters as chunk size grows. The convergence point arises earlier on HPCL (0.5k) than Tardis (6k) due to the difference of network bandwidth between two clusters. This is because in order to reach the maximal network utilization, required chunk size of messages broadcasted on the cluster with slower

network bandwidth is smaller compared to the cluster with faster network bandwidth. Further, we see that similarly on both clusters, pipeline broadcast outperforms binomial tree broadcast after the individual convergence point of pipelining, which complies with Equations 10 as well. Noteworthily, the performance between two types of broadcast differs greater on the cluster with slower network bandwidth (HPCL) than the cluster with faster network bandwidth (Tardis).

6.5 Overall Energy and Performance Efficiency of HP-DAEMON

Experimental results indicate that the optimal energy savings and performance gain can be achieved by applying *HP-DAEMON* in distributed matrix multiplication. We performed two types of experiments to evaluate the effectiveness of *HP-DAEMON* individually: (a) On the energy measurement enabled HPCL cluster, we evaluated energy and performance efficiency of *HP-DAEMON* by comparing ScaLAPACK pdgemm() to our implementation of pdgemm() with and without *HP-DAEMON* (we did not present the data of DPLASMA pdgemm() on HPCL, because DPLASMA pdgemm() did not manifest better performance than ScaLAPACK pdgemm() on Ethernet-switched HPCL); (b) on the Tardis cluster with faster network bandwidth but no tools for energy measurement, we evaluated performance efficiency of our pdgemm() implementation with tuned pipeline broadcast, by comparing against ScaLAPACK, DPLASMA, and our pdgemm() with binomial tree broadcast. The default block size 32 in ScaLAPACK/DPLASMA pdgemm() and the maximal value of N_{grp} (8 in our case) were adopted in our experiments.

Figure 6 shows energy costs and execution time of five different matrix multiplications on the HPCL cluster with two DVFS scheduling strategies, where in our implementation pipeline broadcast with tuned chunk size was employed for achieving the maximal performance of communication. We observe that the time overhead on employing DVFS is non-negligible: 8.1% for ScaLAPACK pdgemm() and 12.4% for our pdgemm() on average. As discussed before, performance loss is attributed to time costs on virtual file read and write operations that are necessary in CPU frequency switching by DVFS. Thus extra energy consumption is incurred during the extra time on frequency switching. Although performance degrades using *Basic DVFS*, overall energy savings are achieved due to the scheduled low-power communication when CPU is idle. Compared to the original versions without DVFS, the energy savings from *Basic DVFS* enabled version of ScaLAPACK pdgemm() and our pdgemm() are considerable 18.1% and 15.1% on average, individually. Compared to *Basic DVFS*, employing *HP-DAEMON* effectively achieves more energy savings and reduces performance loss since the number of DVFS switches is minimized in accordance with the user-specified memory cost threshold. As shown in Figure 6, compared to our pdgemm() without DVFS, more energy savings are fulfilled (22.6% on average and up to 28.8%, 7.5% additional average energy savings compared to *Basic DVFS*) while performance loss is lowered to 6.4% on average (6.0% performance loss is eliminated compared to *Basic DVFS*). The heuristic of reducing frequency switches by grouping is thus evaluated to be beneficial to energy and performance efficiency for distributed matrix multiplication.

As the other integrated part of *HP-DAEMON*, employing pipeline broadcast with tuned chunk size further brings performance gain and thus energy savings. Comparing ScaLAPACK pdgemm() and our pdgemm() both without using DVFS, 32.9% on average and up to 35.6% performance gain is observed. Overall, compared to ScaLAPACK pdgemm(), our pdgemm() with *HP-DAEMON* achieves 47.8% on average and up to 51.4% energy savings, and 28.6% on average and up to 31.5% performance gain, due to the integrated energy and performance efficiency from *HP-DAEMON*. Next we further evaluate overall performance gain achieved by our pdgemm() with *HP-DAEMON* on another cluster with more cores and faster network bandwidth.

Figure 7 shows performance comparison of five other distributed matrix multiplications in

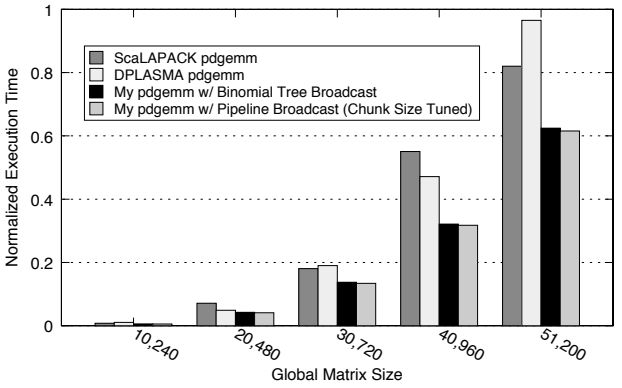


Figure 7: Performance Gain on the Tardis Cluster (512-core, Infiniband).

different implementations on the Tardis cluster. First we see that DPLASMA and ScaLAPACK pdgemm() perform similarly on average. Further the performance difference between binomial tree broadcast and pipeline broadcast narrows down to negligible extent due to the faster communication rate on Tardis. Overall on average, our pdgemm() with pipeline broadcast with tuned chunk size significantly outperforms ScaLAPACK pdgemm() by 33.3% and DPLASMA pdgemm() by 32.7%, respectively. It is thus evaluated that our pdgemm() with *HP-DAEMON* can also be superior in performance efficiency on large-scale clusters with fast network bandwidth.

7 Conclusions

Increasing requirements of exploiting parallelism in high performance applications pose great challenges on improving energy efficiency for these applications. Among potential software-controlled hardware solutions, DVFS has been leveraged to provide substantial energy savings. This paper proposes an adaptive memory-aware DVFS scheduling strategy that reduces the number of CPU frequency switches to minimize the overhead on employing DVFS. A user-specified memory overhead threshold is used for grouping broadcasts/multiplications in distributed matrix multiplication to achieve the optimal energy savings. Further, a pipeline broadcast scheme with tuned chunk size for high performance communication is leveraged to fully exploit network bandwidth. The experimental results on two clusters indicate the effectiveness of the proposed integrated approach in both energy and performance efficiency, compared to ScaLAPACK and DPLASMA matrix multiplication with a basic DVFS scheduling strategy.

8 Acknowledgments

The authors would like to thank the HPCL Lab at the Department of Mathematics, Statistics, and Computer Science of Marquette University for providing the distributed-memory computing cluster with the energy/power profiler PowerPack.

This work is partially supported by US National Science Foundation grants CNS-1118043, CNS-1116691, CNS-1304969, CNS-1305359, and CNS-1305382.

References

[1] *Automatically Tuned Linear Algebra Software (ATLAS).* http://math-atlas.sourceforge.net/.

[2] *DPLASMA: Distributed Parallel Linear Algebra Software for Multicore Architectures.* http://icl.cs.utk.edu/dplasma/.

[3] *ScaLAPACK - Scalable Linear Algebra PACKage.* http://www.netlib.org/scalapack/.

[4] P. Alonso, M. F. Dolz, F. D. Igual, R. Mayo, and E. S. Quintana-Ortí. Saving energy in the LU factorization with partial pivoting on multi-core processors. In *Proc. PDP*, pages 353–358, 2012.

[5] E. Chan, R. van de Geijn, W. Gropp, and R. Thakur. Collective communication on architectures that support simultaneous communication over multiple links. In *Proc. PPoPP*, pages 2–11, 2006.

[6] G. Chen, K. Malkowski, M. Kandemir, and P. Raghavan. Reducing power with performance constraints for parallel sparse applications. In *Proc. IPDPS*, pages 1–8, 2005.

[7] J. Choi. A new parallel matrix multiplication algorithm on distributed-memory concurrent computers. In *Proc. HPC-Asia*, pages 224–229, 1997.

[8] J. Choi, J. J. Dongarra, L. S. Ostrouchov, A. P. Petitet, D. W. Walker, and R. C. Whaley. The design and implementation of the ScaLAPACK LU, QR and Cholesky factorization routines. *Scientific Programming*, 5(3):173–184, August 1996.

[9] A. Faraj and X. Yuan. Automatic generation and tuning of MPI collective communication routines. In *Proc. ICS*, pages 393–402, 2005.

[10] V. W. Freeh and D. K. Lowenthal. Using multiple energy gears in MPI programs on a power-scalable cluster. In *Proc. PPoPP*, pages 164–173, 2005.

[11] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. W. Cameron. PowerPack: Energy profiling and analysis of high-performance systems and applications. *IEEE Trans. Parallel Distrib. Syst.*, 21(5):658–671, May 2010.

[12] C.-H. Hsu and W.-C. Feng. A power-aware run-time system for high-performance computing. In *Proc. SC*, page 1, 2005.

[13] S. Hunold and T. Rauber. Automatic tuning of PDGEMM towards optimal performance. In *Proc. Euro-Par*, pages 837–846, 2005.

[14] N. Kappiah, V. W. Freeh, and D. K. Lowenthal. Just in time dynamic voltage scaling: Exploiting inter-node slack to save energy in MPI programs. In *Proc. SC*, page 33, 2005.

[15] C. Karlsson, T. Davies, C. Ding, H. Liu, and Z. Chen. Optimizing process-to-core mappings for two dimensional broadcast/reduce on multicore architectures. In *Proc. ICPP*, pages 404–413, 2011.

[16] A. Karwande, X. Yuan, and D. K. Lowenthal. CC–MPI: a compiled communication capable MPI prototype for ethernet switched clusters. In *Proc. PPoPP*, pages 95–106, 2003.

[17] H. Kimura, M. Sato, Y. Hotta, T. Boku, and D. Takahashi. Empirical study on reducing energy of parallel programs using slack reclamation by DVFS in a power-scalable high performance cluster. In *Proc. CLUSTER*, pages 1–10, 2006.

[18] D. Li, B. R. de Supinski, M. Schulz, K. W. Cameron, and D. S. Nikolopoulos. Hybrid MPI/OpenMP power-aware computing. In *Proc. IPDPS*, pages 1–12, 2010.

[19] A. Miyoshi, C. Lefurgy, E. V. Hensbergen, R. Rajamony, and R. Rajkumar. Critical power slope: Understanding the runtime effects of frequency scaling. In *Proc. ICS*, pages 35–44, 2002.

[20] B. Rountree, D. K. Lowenthal, S. Funk, V. W. Freeh, B. R. de Supinski, and M. Schulz. Bounding energy consumption in large-scale MPI programs. In *Proc. SC*, pages 1–9, 2007.

[21] R. Springer, D. K. Lowenthal, B. Rountree, and V. W. Freeh. Minimizing execution time in MPI programs on an energy-constrained, power-scalable cluster. In *Proc. PPoPP*, pages 230–238, 2006.

[22] L. Tan, L. Chen, Z. Chen, Z. Zong, R. Ge, and D. Li. Improving performance and energy efficiency of matrix multiplication via pipeline broadcast. In *Proc. CLUSTER*, pages 1–5, 2013.

[23] L. Tan, Z. Chen, Z. Zong, R. Ge, and D. Li. A2E: Adaptively aggressive energy efficient DVFS scheduling for data intensive applications. In *Proc. IPCCC*, pages 1–10, 2013.

[24] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *Proc. OSDI*, page 2, 1994.